Gibbs_sampling_GMM

August 3, 2020

1 Inference of Gaussian Mixture Models using Gibb's sampling

Please refer to Gibb's sampling pdf attached in the same directory titled "LR_inference_via_gibbs_sampling" to get a summary of Gibb's sampling. Here, we will look at Gaussian Mixture Model (GMM) and derive equations to infer the parameters.

1.1 GMMs

A GMM is mixture of several gaussians each with their means (defining the center), standard deviation or covariance (defining the width or area of the distribution) and a mixing weight (defining the probability of a random variable belonging to any distribution).

Let us take the example of average height of male and female in different countries. I have uploaded a csv file titled "heights" in the same repository. Let us plot the data from this file.

[1]:
```
import pandas as pd
import numpy as np
import random
import scipy.stats
import matplotlib.pyplot as plt

%matplotlib inline
```

[2]:
```
# Using pandas dataframe to read and store the csv file

df_ht = pd.read_csv("~/heights.csv")
df_ht
```

[2]:
```
         country  maleMetricHeight  femaleMetricHeight maleImperialHeight  \
0          Samoa               NaN               166.6                NaN
1     Montenegro             183.4               169.4         6 ft 0 in
2        Iceland             181.0               168.0     5 ft 11 1/2 in
3        Comoros               NaN               154.8                NaN
4      Swaziland               NaN               159.1                NaN
..           ...               ...                 ...                ...
117    Sri Lanka             163.6               151.4      5 ft 4 1/2 in
118  Philippines             163.5               151.8      5 ft 4 1/2 in
119        Nepal             163.0               150.8          5 ft 4 in
```

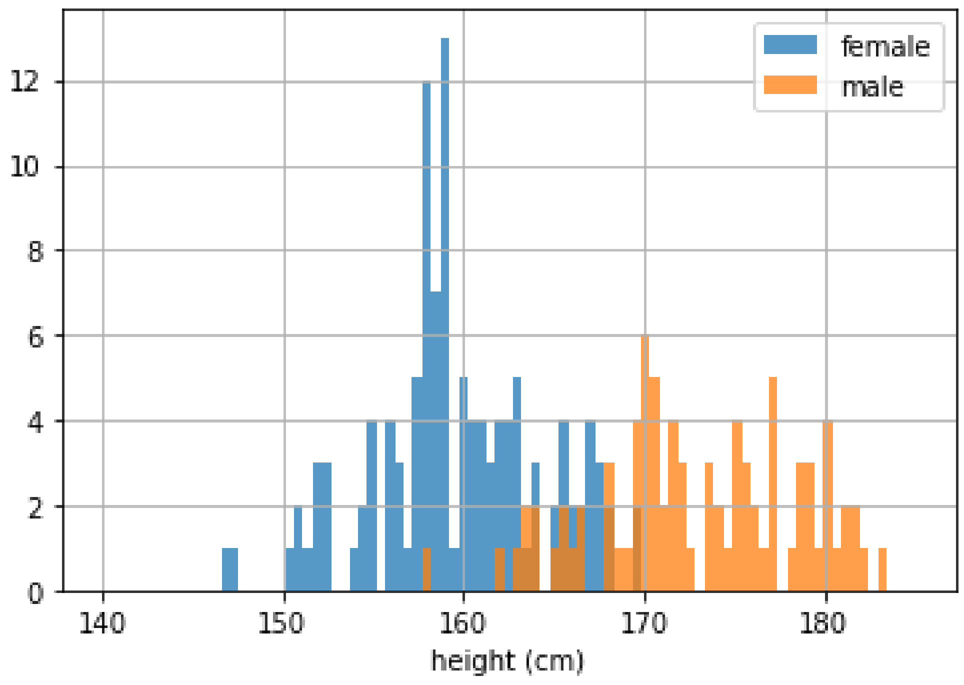
```
120       Vietnam              162.1               152.2           5 ft 4 in
121     Indonesia              158.0               147.0           5 ft 2 in

    femaleImperialHeight     pop2020
0          5 ft 5 1/2 in     198.414
1          5 ft 6 1/2 in     628.066
2              5 ft 6 in     341.243
3              5 ft 1 in     869.601
4          5 ft 2 1/2 in    1160.164
..                   ...         ...
117       4 ft 11 1/2 in   21413.249
118            5 ft 0 in  109581.078
119       4 ft 11 1/2 in   29136.808
120            5 ft 0 in   97338.579
121           4 ft 10 in  273523.615

[122 rows x 6 columns]
```

[3]:

```
# Replacing NaN with the mean of that column

df_ht.fillna(df_ht.mean())
```

[4]:

```
# Plot a histogram of the data

bins = np.linspace(140, 185, 90)

plt.figure()
plt.hist(df_ht.femaleMetricHeight, bins, alpha=0.75, label='female')
plt.hist(df_ht.maleMetricHeight, bins, alpha=0.75, label='male')
plt.legend()
plt.xlabel("height (cm)")
plt.grid()
plt.show()
```



In the histogram plotted above, we are observing $x_1, ..., x_n$ where each random variable x_i is a sample belonging to either the male dataset or the female. There are two labels ($K = 2$) here, male and female which are associated with x_i. This label is called latent variable and is denoted by Z_k where $k \in 1, ..., K$.

According to law of total probability,

$$P(x_i = x) = \sum_{k=1}^{K} P(x_i = x | Z_i = k) P(Z_i = k)$$

where $P(Z_i = k)$ is called mixture weight and is denoted by π_k. All the K mixture weights sum upto 1.

[5]:
```
# True parameter values of the female distribution
df_ht.femaleMetricHeight.mean(), df_ht.femaleMetricHeight.std()
```

[5]:
```
(159.67075630252103, 4.699890759578636)
```

[6]:
```
# True parameter values of the female distribution
df_ht.maleMetricHeight.mean(), df_ht.maleMetricHeight.std()
```

[6]:
```
(172.9321686746988, 5.5694132125545766)
```

[7]:
```
# Create a dataset with only heights from the dataframe and shuffle them
data = list(df_ht.femaleMetricHeight)+list(df_ht.maleMetricHeight)
random.shuffle(data)
```

```
# Filter out outliers - the entires that are 0
print("Length of the dataset before removing outliers:",len(data))
data = list(filter(lambda d: d>=1e-2,data))
print("Length of the dataset after removing outliers: ",len(data))
```

```
Length of the dataset before removing outliers: 244
Length of the dataset after removing outliers:  202
```

[8]:
```
# Find the mean and std dev. of the entire dataset

mu_0    = sum(data)/len(data)
sigma_0 = np.std(np.array(data))
print(mu_0,sigma_0)
```

```
165.1197524752475 8.25014535954264
```

1.2 Deriving the update equations for Gibb's sampling

Gibb's sampling for a GMM consists of the following,

Chose random starting means for both male and female heights (`mus = [mu_f,mu_m]`), a random mixing weights vector (`pis = [pi_f,pi_m]`) and a randomly assign each of the height data points with a label (`z`).

1. Chose a sampling order, here I have chosen to update `pis`, `z` and then `mus`.
2. Sample the conditional probabilities of the parameters with the updated other parameters.

Continue until convergence

We wish to place conjugate priors for the parameters so that it is easier to find the posterior probability. Let,

1. $\mu_f \sim \mathcal{N}(\mu_{f0}, 1/\tau_{f0})$
2. $\mu_m \sim \mathcal{N}(\mu_{m0}, 1/\tau_{m0})$

General

Let us assume there is variable x that is normally distributed with mean μ and precision τ,

$$\begin{aligned}
f(x|\mu, 1/\tau) &= \sqrt{\frac{\tau}{2\pi}}\, e^{-\frac{\tau}{2}(x-\mu)^2} \\
&= \text{const} \cdot e^{(-\frac{\tau x^2}{2} - \frac{\tau\mu^2}{2} - \tau x\mu)} \\
ln(f(x|\mu, 1/\tau)) &= ln(\text{const}) + ln(e^{(-\frac{\tau x^2}{2} - \frac{\tau\mu^2}{2} - \tau x\mu)}) \\
&= ln(\text{const}) - \frac{\tau x^2}{2} - \frac{\tau\mu^2}{2} + \tau x\mu \\
&= \frac{\tau x^2}{2} + \tau x\mu \quad \text{(drop terms that are independent of x)}
\end{aligned}$$

Having derived this, note that,

1. The coefficient of x^2 is $\frac{-\tau}{2}$
2. The coefficient of x is $\tau\mu$

Deriving μ We know from Bayes rule that,

$$\begin{aligned}
p(\mu|x,z,\pi,1/\tau) &\propto p(X|\mu,z,\pi,1/\tau)p(\mu|z,\pi,1/\tau)\\
&\propto \prod_{i=1}^{N}\sum_{k=1}^{K}\frac{\tau}{\sqrt{2}\pi}e^{-\frac{\tau}{2}(x-\mu_k)^2}\cdot\frac{\tau_0}{\sqrt{2}\pi}e^{-\frac{\tau_0}{2}(\mu-\mu_0)^2}\\
ln(p(\mu|x,z,\pi,1/\tau)) &\propto \sum_{i=1}^{N} ln[\sum_{k=1}^{K}\frac{\tau}{\sqrt{2}\pi}e^{-\frac{\tau}{2}(x-\mu_k)^2}] + ln(\frac{\tau_0}{\sqrt{2}\pi}) - \frac{\tau_0}{2}(\mu-\mu_0)^2
\end{aligned}$$

Let us focus on μ_f and drop terms that do not have it

$$\begin{aligned}
ln(p(\mu_f|x,z_f,\pi,1/\tau)) &\propto \sum_{i=1}^{N_f}[ln(\frac{\tau}{\sqrt{2}\pi}) - \frac{\tau x^2}{2} - \frac{\tau\mu_f^2}{2} + \tau x\mu_f] + ln(\frac{\tau_{f0}}{\sqrt{2}\pi}) - \frac{\tau\mu^2}{2} - \frac{\tau\mu_{f0}^2}{2} + \tau\mu_{f0}\mu\\
&\text{(drop all the terms that do not contain } \mu_f \text{ and simplifying)}\\
&\propto -\frac{\tau}{2}\mu_f^2 N_f + \tau\mu_f\sum_{i=1}^{N_f}x_i - \frac{\tau_{f0}}{2}\mu_f^2 + \tau_{f0}\mu_f\mu_{f0}
\end{aligned}$$

Having derived this, note that,

1. The coefficient of μ_f^2 is $\frac{-\tau}{2}N_f - \frac{\tau_{f0}}{2}$
2. The coefficient of μ_f is $\tau\sum_{i=1}^{N_f}x_i + \tau_{f0}\mu_{f0}$

Similarly, for μ_m,

1. The coefficient of μ_m^2 is $\frac{-\tau}{2}N_m - \frac{\tau_{m0}}{2}$
2. The coefficient of μ_m is $\tau\sum_{i=1}^{N_m}x_i + \tau_{m0}\mu_{m0}$

So, μ_f and μ_m look Gaussian and matching their coefficients with that of variable x derivation from earlier and calculating the mean and precision, we define the conditional density

$$\begin{aligned}
\mu_f &\sim \mathcal{N}(\frac{\tau_{f0}\mu_{f0} + \tau\sum_{i=1}^{N_f}(x_i)}{\tau_{f0}+\tau N_f}, \frac{1}{\tau_{f0}+\tau N_f})\\
\mu_m &\sim \mathcal{N}(\frac{\tau_{m0}\mu_{m0} + \tau\sum_{i=1}^{N_m}(x_i)}{\tau_{m0}+\tau N_m}, \frac{1}{\tau_{m0}+\tau N_m})
\end{aligned}$$

Let us now write this in python

[9]:
```
def sample_mus(data,z,tau,mu_0,tau_0):
    N     = len(data)
    assert len(z) == N

    N_k   = [N-sum(z),sum(z)]

    precision_f = tau_0[0] + tau*N_k[0]
    precision_m = tau_0[1] + tau*N_k[1]

    mean_f = ((tau_0[0]*mu_0[0]) + tau*sum([data[i] for i in range(N) if␣
 ↪z[i]==0])) / precision_f
    mean_m = ((tau_0[1]*mu_0[1]) + tau*sum([data[i] for i in range(N) if␣
 ↪z[i]==1])) / precision_m

    return [np.random.normal(mean_f, 1/np.sqrt(precision_f)),np.random.
 ↪normal(mean_m, 1/np.sqrt(precision_m))],N_k
```

Deriving z Conditional distribution of z is,

$$\begin{aligned} p(z_i|x_i,\pi_i,\mu_i,1/\tau) &\propto p(x_i|z_i,\pi_i,\mu_i,1/\tau)p(z_i|\pi_i,\mu_i,1/\tau) \\ &\propto \text{pdf}(x_i)\cdot\pi_{z_i} \end{aligned}$$

[10]:
```
def sample_z(data,pi,mus,tau):

    x_minus_mu          = [(d - mus[0],d - mus[1]) for d in data]
    post_z_given_x      = [(pi[0]*scipy.stats.norm(0,1/np.sqrt(tau)).pdf(f) , \
                            pi[1]*scipy.stats.norm(0,1/np.sqrt(tau)).pdf(m)) for␣
 ↪(f,m) in x_minus_mu]
    norm_post_z_given_x = [(f/(f+m),m/(f+m)) for (f,m) in post_z_given_x]
    z                   = [0 if f > m else 1 for (f,m) in norm_post_z_given_x]

    return z
```

Deriving π A Dirichlet distribution is often used to probabilistically categorize events among several categories. We shall employ the same here.

$$p(\pi_k|x,z,\mu,1/\tau) = \mathcal{D}ir(\alpha + N_k)$$

[11]:
```
def sample_pi(N_k,alpha):
    return np.random.dirichlet(alpha + np.array(N_k))
```

Now that we have successfully derived all the update equations, let us define the hyperparameters. Through Gibb's sampling, our aim is to look at only the height data not knowing if it is a male height or female and to infer their means and the mixing weights. I am assuming a constant precision `tau` for this data.

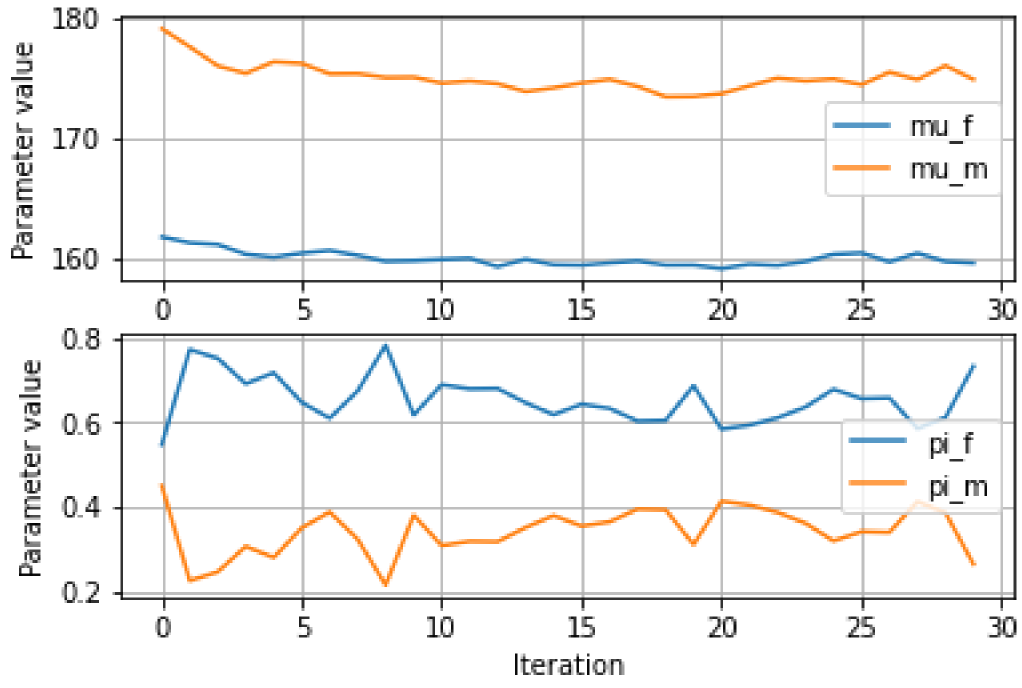
[39]:
```
# Specify hyper parameters
hyper  = {"tau": 1/4,
          "mu_f0":150,
          "tau_f0":1/16,
          "mu_m0":200,
          "tau_m0":1/16,
          "alpha": 2,
          "K": 2}
```

[40]:
```
def gibbs_sampling(data,iters,hyper,z):
    assert len(data) == len(z)
    N = len(data)

    mus = [hyper["mu_f0"],hyper["mu_m0"]]

    trace = np.zeros((iters, 4))
    N_k = [N-sum(z),sum(z)]

    for it in range(iters):
        pi          = sample_pi(N_k,hyper["alpha"])
        z           = sample_z(data,pi,mus,hyper["tau"])
        mus, N_k    = sample_mus(data,z,hyper["tau"],
        [hyper["mu_f0"],hyper["mu_m0"]],[hyper["tau_f0"],hyper["tau_m0"]])

        trace[it,:] = np.array(mus+list(pi))

    trace = pd.DataFrame(trace)
    trace.columns = ['mu_f', 'mu_m', 'pi_f', 'pi_m']
    return trace
```

[41]:
```
iters = 30

# Assign a random label to each height
z = [random.randrange(0,2) for i in range(len(data))]

# Run Gibb's sampling
trace = gibbs_sampling(data,iters,hyper,z)
```

[45]:
```
# Plot the trace

fig, axs = plt.subplots(2)
axs[0].plot(trace.loc[:,["mu_f","mu_m"]])
```

```
axs[0].legend(["mu_f","mu_m"])
axs[0].set_xlabel("Iteration")
axs[0].set_ylabel("Parameter value")
axs[0].grid()

axs[1].plot(trace.loc[:,["pi_f","pi_m"]])
axs[1].legend(["pi_f","pi_m"])
axs[1].set_xlabel("Iteration")
axs[1].set_ylabel("Parameter value")
axs[1].grid()
plt.show()
```



[54]:
```
mu_f_burnin = trace.loc[20:,["mu_f"]].mean()[0]
mu_m_burnin = trace.loc[20:,["mu_m"]].mean()[0]

print("Average female height in the world: {:3.4f}".format(mu_f_burnin))
print("Average male height in the world:   {:3.4f}".format(mu_m_burnin))
```

```
Average female height in the world: 159.9048
Average male height in the world:   174.9064
```

So, thats's it! We have successfully inferred the means from the heights data.